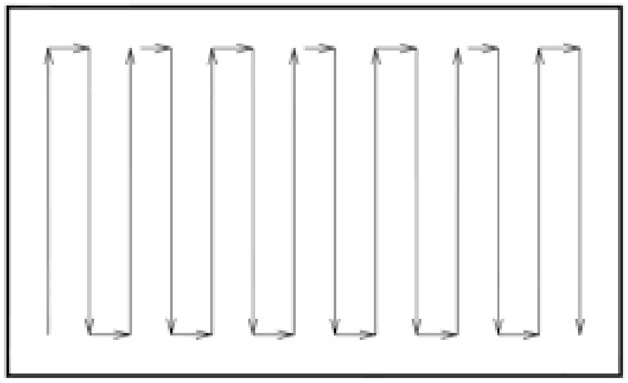
9. Mauricio

Program Name: Mauricio.java Input File: mauricio.dat

Mauricio's school just purchased a robotic lawnmower to mow all the rectangular patches of grass on their campus. Examples include the football field, soccer field, and the playground. The robotic lawnmower does not have an efficient path planning algorithm however, and will just roam around randomly until the rectangular region is fully covered. Mauricio's principal has asked him to implement a boustrophedon path plan to make the lawnmower more efficient in both time and energy. The Greek term boustrophedon translates to "the way of the ox". Today, boustrophedon means from right to left and from left to right in alternate lines. As the term implies the path plan requires the robot to traverse the full length of the field, turn around 180 degrees, mow the next portion of uncut grass, traverse back the full length of the field, turn around 180 degrees, and continue. This process is completed until the full area is covered. The below figure gives a visual example of a boustrophedon path plan.



Mauricio noticed that he could write an even better boustrophedon path plan that traversed the longest edge of rectangle first, versus the shortest edge. In traversing the longest edge first, his robot would not have to turn 180 degrees as much, reducing both time and energy! Can you help Maurico write a path planning program that implements a boustrophedon path plan that traverses the longest edge first?

Input: Input starts with a line containing an integer N (1 <= N <= 10), the number of test cases. The following N lines, each with two integer values representing the width W (number of rows) and the length L (number of columns) of the rectangular region to mow. The constraints for W and L are as follows:

$$2 \leq W \leq 70$$
$$2 \leq L \leq 70$$
$$L \neq W$$

Output: For each width W and length L, you are to print out the corresponding boustrophedon path plan that traverses the longest edge of the rectangular matrix first. Your path plan will incrementally number the order of which the elements of the rectangular regions are to be visited. Your output should be printed out in columns of D+1 size and right justified, where D is the number of digits of the last element visited. For example, if the last element visited was numbered 120, D would equal 3, so your output should be in columns of size 4 (D+1). Output numbering begins at 1, and always begins in the top, left element. Following each output set should be a line of 10 (ten) equal signs.

Sample input:

```
4
2 3
3 2
12 10
10 12
```

(continued next page)

Mauricio – continued

Sample output:
```
  1  2  3
  6  5  4
==========
  1  6
  2  5
  3  4
==========
    1   24   25   48   49   72   73   96   97  120
    2   23   26   47   50   71   74   95   98  119
    3   22   27   46   51   70   75   94   99  118
    4   21   28   45   52   69   76   93  100  117
    5   20   29   44   53   68   77   92  101  116
    6   19   30   43   54   67   78   91  102  115
    7   18   31   42   55   66   79   90  103  114
    8   17   32   41   56   65   80   89  104  113
    9   16   33   40   57   64   81   88  105  112
   10   15   34   39   58   63   82   87  106  111
   11   14   35   38   59   62   83   86  107  110
   12   13   36   37   60   61   84   85  108  109
==========
    1    2    3    4    5    6    7    8    9   10   11   12
   24   23   22   21   20   19   18   17   16   15   14   13
   25   26   27   28   29   30   31   32   33   34   35   36
   48   47   46   45   44   43   42   41   40   39   38   37
   49   50   51   52   53   54   55   56   57   58   59   60
   72   71   70   69   68   67   66   65   64   63   62   61
   73   74   75   76   77   78   79   80   81   82   83   84
   96   95   94   93   92   91   90   89   88   87   86   85
   97   98   99  100  101  102  103  104  105  106  107  108
  120  119  118  117  116  115  114  113  112  111  110  109
==========
```